(m, k)–firm constraints and DBP scheduling: impact of the initial k-sequence and exact schedulability test

Joël Goossens
Université Libre de Bruxelles (U.L.B.)
Brussels, Belgium
joel.goossens@ulb.ac.be

September 8, 2008

Abstract

In this paper we study the scheduling of (m, k)–firm synchronous periodic task systems using the Distance Based Priority (DBP) scheduler. We first show three phenomena: (i) choosing, for each task, the initial k-sequence 1^k is not optimal, (ii) we can even start the scheduling from a (fictive) error state (in regard to the initial k-sequence) and (iii) the period of feasible DBP-schedules is not necessarily the task hyper-period. We then show that any feasible DBP-schedule is periodic and we upper-bound the length of that period. Lastly, based on our periodicity result we provide an exact schedulability test.

Keywords. (m, k)–firm constraints, real-time scheduling, uniprocessor, periodic tasks, non-preemptive tasks.

1 Introduction

In this paper we consider the scheduling of (m, k)–firm real-time periodic task sets. The (m, k)–firm model was initially introduced by Hamdaoui et al. [1]; this model is intermediary between *hard* real-time constraints (where deadline misses are fatal) and *soft* real-time constraints (where deadline misses are tolerated but minimized). Indeed, the (m, k)–firm constraint imposes that "few" request deadlines can be missed. More formally, for each task, the (m, k)–firm constraint is characterized by two parameters: m and k, and the constraint requires that at least m requests meet their deadline for any k consecutive requests of the task. This model is very often used to handle the scheduling of messages on real-time networks, the tasks represent the handling of network frames and are consequently and inherently *non-preemptive* in the model. Hence, in the following, the term task denotes either a non-preemptive task (scheduled on a *uni*processor) or a message (scheduled on a network channel). The seminal paper of Hamdaoui et al. [1] defined also a scheduling algorithm for that kind of constraints: the Distance Based Priority (DBP in short). Under DBP, the priority of each task is dynamic and is based on the number of task request[s] which can miss their deadline without violating the (m, k)–firm constraint.

Related research. Since the seminal paper of Hamdaoui et al. [1], there is a huge literature about scheduling firm real-time systems for various kinds of schedulers. It is out of the scope of this paper to summarize that literature. Our study concerns a specific task model ((m, k)–firm periodic tasks) for a specific scheduler (DBP). Concerning the more specific case of (m, k)–firm constraints and the scheduler DBP, the literature proposes sufficient schedulability conditions [4]. Hamdaoui in [2] evaluated the probability failure; Lindsay et al. [5] extend the analysis to handle point-to-point networks; Stringel et al. [8] handle multihop networks; Poggi et al. [6] introduce

matrix-DBP scheduling and show the improvement. Related particular cases include the work of Jeffay et al. [3] and the work of Quan et al. [7].

In all those papers, the choice/value of the initial k-sequences[1] used by DBP to start the scheduling is not really studied. There is no discussion in the literature about the value of the initial k-sequence. Very often, the researchers seem to assume that choosing the pattern[2] 1^k for those sequences is optimal regarding the system schedulability without any discussion. Authors also consider very often, e.g., in experimental studies, that examining the first hyper-period of the schedule is significant to conclude. Lastly, the literature, to the best of our knowledge, only proposes sufficient (or necessary) schedulability tests.

This research. In this research we show that the choice of the initial k-sequences is significant and that the period of the schedule can be larger than the hyper-period. We first show three phenomena: (i) choosing, for each task, the initial k-sequence 1^k is not optimal, (ii) we can even start the scheduling from a (fictive) error state (in regard to the initial k-sequence) and (iii) the period of feasible DBP-schedules is not necessarily the task hyper-period. We then show that any feasible DBP-schedule is periodic and we upper-bound the length of that period. Lastly, based on our periodicity result we provide an exact schedulability test. From the best of our knowledge, this is the first *exact* (i.e., necessary and sufficient) such test.

Organization. Section 2 presents our model of computation, preliminary definitions and assumptions. Section 3 presents the three (counter-intuitive) important phenomena, generally ignored by the literature, concerning the initial k-sequences. Section 4 studies the periodicity of DBP schedules. Section 5 provides a feasibility interval and, based on our periodicity result, our exact schedulability test. Lastly, in Section 6, we conclude.

2 Definitions and assumptions

We consider the scheduling of synchronous periodic task systems. A system τ is composed by n periodic tasks $\tau_1, \tau_2, ..., \tau_n$, each task is characterized by a period T_i, a relative deadline D_i, an execution requirement C_i. Such a periodic task generates an infinite sequence of jobs, with the k^{th} job arriving at time-instant $(k-1)T_i$ ($k = 1, 2, ...$), having an execution requirement of C_i units, and a deadline at time-instant $(k-1)T_i + D_i$. We consider constrained-deadline systems, i.e., $D_i \leq T_i$. Two additional task characteristics are m_i and k_i which mandate that at least m_i out of any k_i consecutive jobs of τ_i must meet its deadline. We consider in this paper a discrete model, i.e., the characteristics of the tasks and the time are integers.

In order to schedule dynamically such a system, the scheduler will typically based its decision on the k-sequence for each task, with the following definition:

Definition 1 (k-sequence)**.** *The k-sequence of task τ_i is a binary string $W = [w_{i,1}, w_{i,2}, ..., w_{i,k_i}]$ which represents the recent past of the task jobs. By definition a deadline miss corresponds to the value '0' and a deadline met to the value '1', the leftmost bit represents the oldest job.*

Notice that at time 0, this notion of k-sequence is not really defined since the "past" is somewhat undefined, or empty. We assume that initially, at time 0, the scheduler based its decision on an *initial* k-sequence for each task; generally the authors consider the string 1^{k_i} for task τ_i, but we will see that is not optimal and causes a loss of generality.

The algorithm DBP assigns a dynamic priority to each active task (say τ_i) based on the number of task request[s] which can miss their deadline without violating the (m_i, k_i)–firm constraint. In other words, DBP bases its decision on the *distance* between the current state (k_i-sequence)

[1] Informally, for each task, the k-sequence represents the recent past of the task in regard with deadline failures. See Definition 1 for a formal definition.

[2] In this document we use the formal language notation, w^α means w repeated α times.

	T_i	C_i	m_i	k_i
τ_1	4	1	2	4
τ_2	10	8	3	4

Table 1: System characteristics.

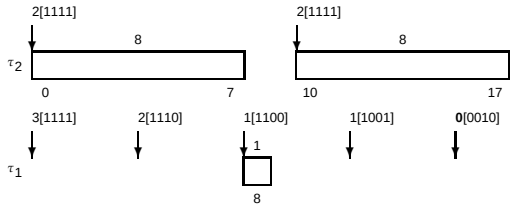

Figure 1: System not DBP-schedulable choosing the initial k-sequence 1^k: at time 16, τ_1 reaches an error state, the $(2, 4)$–firm constraint of τ_1 being violated.

and the closest error state (i.e., where the number of '1' is less than m_i). For instance, if τ_i is subject to a $(2, 3)$–firm constraint and if the current 3-sequence is [101], the distance is 1 (the current job of τ_i must be executed) while if the current 3-sequence is [011] the distance is 2 (the current request of τ_i could be executed, but if not the next one must be). By definition, any k-sequence with less than m_i '1' violates the (m_i, k_i)–firm constraint and corresponds to an error state.

DBP assigns the highest priority to the active task with the smallest distance; variants of DBP are based on the additional rule used to break ties, e.g., EDF-DBP or RM-DBF if we use the Earliest Deadline First or the Rate Monotonic scheduler. We do not consider a specific tie-broker in this study but we assume that the tie-broker is deterministic and memoryless with the following definitions:

Definition 2 (Deterministic algorithm)**.** *A scheduling algorithm is said to be* deterministic *if it generates a unique schedule for any given set of jobs.*

Definition 3 (Memoryless algorithm)**.** *A non-preemptive scheduling algorithm is said to be* memoryless *if the scheduling decision made by it at time t (which corresponds to job arrival or completion) depends only on the static characteristics of active tasks (i.e., T_i, C_i, m_i, k_i) and on the current state of the system (i.e., the current k-sequence and the time elapsed since the last request, of each task).*

In the previous definition, it may be noticed that since we consider *non-preemptive* systems, the scheduler does not consider the remaining processing time, the latter is always equal to C_i for any active task τ_i at scheduling time.

3 Study of the initial k-sequences

3.1 The non-optimality of the string 1^{k_i}

In this section we will see that choosing, for each task (say τ_i), the initial k-sequence to be 1^{k_i}, i.e., consider that the k_i previous requests completed by their deadline is not optimal and causes a loss of generality.

Observation 1 (Non-optimality of 1^*)**.** *Choosing the pattern 1^{k_i} for the initial k-sequence of each periodic task τ_i is not optimal under DBP.*

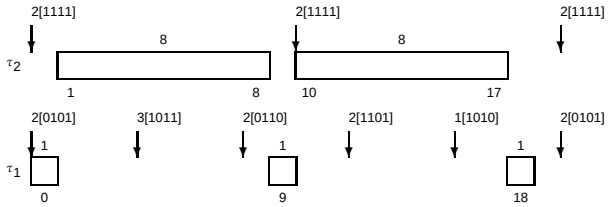

Figure 2: The system is DBP-schedulable since all (m, k)–firm constraints are met in $[0, 20)$ and the system is in the same state at time 20, than at time 10.

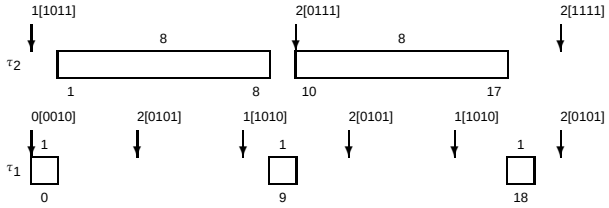

Figure 3: The system is DBP-schedulable since all (m, k)–firm constraints are met in $[0, 20)$ and the fact that the system is in the same state at time 20 than at time 0 in Figure 2.

Proof. The proof is based on a (counter-)example which exhibits a system which is not DBP-schedulable choosing the pattern 1^{k_i} for the initial k-sequence of each periodic task τ_i while the system is DBP-schedulable for another initial k-sequences. Our counter-example is described by Table 1. Figure 1 shows[3] that the system is not DBP-schedulable if we start to schedule the system with the k-sequence 1^{k_i} for each task τ_i. (Notice that Figure 1 corresponds also to both RM-DBP and EDF-DBP schedules.)

On the contrary, with the following k-sequences: $[0101], [1111]$ for τ_1 and τ_2, respectively, the system is DBP-schedulable as illustrated by Figure 2. (Notice that Figure 2 corresponds to both RM-DBP and EDF-DBP schedules.) □

3.2 Starting from an error state

In the previous section we saw that starting with the k-sequence 1^k is not always optimal, we also show, with the next example, that we can even start from an error state.

Example 1. *We consider this time the very same periodic task system than the one considered in the previous section (i.e., defined by Table 1) but we consider the following initial k-sequences:* $[0010], [1011]$ *for τ_1 and τ_2, respectively. Although we consider an initial fallacious k-sequence for τ_1 (since* $[0010]$ *includes a single '1'), Figure 3 shows that the system is DBP-schedulable: all* (m, k)*–firm constraints are met (since they are met in* $[0, 20)$*, and since at time 20 we reach the same system state than the schedule illustrated by Figure 2 (at time 0), consequently the feasible schedule repeats from time 20).*

We leave open the question of choosing efficiently/optimally the initial k-sequences and we assume in the following that those sequences are already chosen but are possibly different from 1^{k_i}.

[3] In our figures, picture ↓ represents a task request, we denote by $d[W]$ the corresponding k-sequence W and the current distance d from the closest error state.

	T_i	C_i	m_i	k_i
τ_1	3	2	1	3
τ_2	3	2	1	4

Table 2: System characteristics.

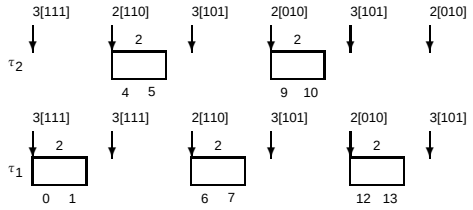

Figure 4: The system is DBP-schedulable since all (m, k)–firm constraints are met in $[0, 15)$ and the system is in the same state at time 15 than at time 9.

4 The periodicity of DBP-schedules

In this section we will study the periodicity of DBP-schedules, an instrumental property to design an exact schedulability test.

It is quite obvious that any feasible (deterministic and memoryless) schedule is periodic (we will show that property below). Previous examples show that the periodic part of the schedule does not necessarily start from the origin (e.g., Figure 3). Another important phenomenon is the fact that the period of the schedule is not necessarily equal to the hyper-period $P \stackrel{\text{def}}{=} \text{lcm}\{T_i \mid i = 1, \ldots, n\}$ as exhibited by the next example.

Example 2. *Consider the task characteristics described by Table 2. Figure 4 shows that the system is DBP-schedulable and that the period of the schedule is* twice *the hyper-period (3). The schedule repeats from time 6 but the period is 6.*

Now we will prove the schedule periodicity and (upper-)bound the length of that period. That result (and its proof) is instrumental to design our *exact* schedulability test for the DBP algorithm (see Algorithm 1).

Theorem 1 (Period upper-bound)**.** *Any feasible DBP-schedule of a synchronous* (m, k)*–firm periodic task set is finally periodic. Moreover, the period of the schedule is a multiple of* P*, upper-bounded by* $\prod_{i=1}^{n} \sum_{j=m_i}^{k_i} \binom{k_i}{j} \times P$*.*

Proof. DBP is an on-line scheduling algorithm, i.e., it takes its decisions based on the (static) task characteristics and the (dynamic) current state of the system: the current k-sequence and the time elapsed since the last request, of each task. Since we consider *synchronous* periodic, *constrained-deadline*, and DBP-feasible task systems, we know that at each instant $t = k \cdot P$ ($k \in \mathbb{N}$) all requests which occur strictly before t have completed their execution. Moreover, a new request of each task occurs at time t. Consequently, regarding the time elapsed since the last request, the system is in the same state at time $0, P, 2P \ldots$. The only difference (if any) concerns the k-sequences. For a given task (say τ_i) we can distinguish between $\sum_{j=m_i}^{k_i} \binom{k_i}{j}$ distinct k-sequences (i.e., k-sequences with at least m_i '1' of length k_i). Consequently, in the worst case we have to consider $\prod_{i=1}^{n} \sum_{j=m_i}^{k_i} \binom{k_i}{j}$ hyper-periods before reaching a state already considered in the past. The property follows from the fact we consider a deterministic and memoryless scheduler. □

5 Exact schedulability test

Now we have the material to provide an exact schedulability test (and an algorithm) for any deterministic and memoryless DBP scheduler. Notice that we assume that the initial k-sequences are fixed.

5.1 Feasibility interval

Definition 4 (Feasibility interval)**.** *A* feasibility interval *is a finite interval such that if no* (m,k)*–firm constraint is missed while considering only requests within this interval then no* (m,k)*–firm constraint will ever be missed.*

Corollary 2. *The interval* $\left[0, \prod_{i=1}^{n} \sum_{j=m_i}^{k_i} \binom{k_i}{j} \times P\right)$ *is a feasibility interval for the scheduling of synchronous constrained-deadline* (m,k)*–firm periodic systems using DBP.*

Proof. It is a direct consequence of the proof of Theorem 1: in the worst case the system state at time $0, P, 2P, \left[\left(\prod_{i=1}^{n} \sum_{j=m_i}^{k_i} \binom{k_i}{j}\right) - 1\right] P$ are different but the system state at time $\prod_{i=1}^{n} \sum_{j=m_i}^{k_i} \binom{k_i}{j} \times P$ by Theorem 1 occurred already in the past. □

5.2 Exact Algorithm

Based on Corollary 2 a straightforward exact schedulability test consists in building the schedule (by means of simulation) in the time interval $[0, \prod_{i=1}^{n} \sum_{j=m_i}^{k_i} \binom{k_i}{j} \times P)$. We designed a *faster* (in average) algorithm (Algorithm 1) which stops once the schedule repeats (or a failure occurs). The idea is to compare, at each hyper-period, the system state with the previous hyper-period states. Algorithm 1 assumes that the function **Schedule** stops the simulation and returns false once a (m,k)–firm constraint is violated. The **system-state** considered in Algorithm 1 is the current k-sequence for each task.

The worst-case time complexity of the exact schedulability test is $O(\prod_{i=1}^{n} \sum_{j=m_i}^{k_i} \binom{k_i}{j} \times P)$. We believe that for many real real-time applications the time complexity of our *off-line* test is reasonnable, e.g., if we consider harmonic task periods and limited (m,k) parameters.

6 Conclusion

In this study we showed that the choice of the initial k-sequences *is* significant regarding the system schedulability. We exhibited three phenomena: (i) choosing, for each task, the initial k-sequence 1^k is not optimal, (ii) we can even sometimes start the scheduling from a (fictive) error state and (iii) the period of feasible DBP-schedules is not necessarily the task hyper-period. We then showed that any feasible DBP-schedule is periodic and we upper-bounded the length of that period. Lastly, based on our periodicity result we provided an exact schedulability test.

Future work. We left open the question of choosing efficiently/optimally the initial k-sequences. We left also open the following question: does the initial k-sequence impact the period of the (feasible) schedule? This research could be also extended to consider sporadic or asynchronous periodic task sets.

Acknowledgments. The author would like to thank Qiong Ye Song and Raymond Devillers for taking part in interesting discussions. Finally, comments of anonymous reviewers helped improving the presentation of the paper.

Algorithm 1: Exact DBP-schedulability test.

```
Input: task set τ
Output: feasible
Data: State St[max]
/* max = ∏_{i=1}^{n} Σ_{j=m_i}^{k_i} (k_i choose j)                    */
/* array St is indexed from 0                                          */
begin
    /* we save the current system state                                */
    St[0] := system-state ;
    ℓ := 0;
    current-time := 0 ;
    periodic := false ;
    feasible := true ;
    repeat
        feasible := Schedule from current-time to current-time + P ;
        current-time := current-time + P ;
        if ℓ = max –1 then
            for j := 0 to max – 2 do
                St[j] := St[j + 1];
        else
            ℓ := ℓ + 1;
        St[ℓ] := system-state ;
        j := ℓ – 1 ;
        while (j > 0 ∧ ¬periodic) do
            periodic := (St[j] = St[ℓ]) ;
            j := j – 1;
    until (periodic ∨ ¬feasible) ;
    return feasible;
end
```

References

[1] Hamdaoui, M., and Ramanathan, P. A dynamic priority assignment technique for streams with (m, k)–firm deadlines. *IEEE Transactions on Computers 44*, 12 (December 1995), 1443–1451.

[2] Hamdaoui, M., and Ramanathan, P. Evaluating dynamic failure probability for streams with (m, k)–firm deadlines. *IEEE Transactions on Computers 46*, 12 (December 1997), 1325–1337.

[3] Jeffay, K., Stanat, D., and Martel, C. On non-preemptive scheduling of periodic and sporadic tasks. In *Real-Time Systems Symposium* (1991).

[4] Li, J., Song, Y. Q., and Simonot-Lion, F. Schedulability analysis for systems under (m, k)–firm constraints. In *Factory Communication Systems* (September 2004), IEEE Computer Society, pp. 23–30.

[5] Lindsay, W., and Ramanathan, P. Dbp-m: a technique for meeting end-to-end (m, k)–firm guarantee requirements in point-to-point networks. In *Annual conference on local computer networks* (1997), I. C. Society, Ed., pp. 294–303.

[6] Poggi, E., Song, Y. Q., Koubaa, A., and Wang, Z. Matrix-dbp for (m, k)–firm real-time guarantee. In *Real-Time and Embedded Systems* (Paris, France, 2003), Y. Trinquet, Ed., pp. 457–480.

[7] Quan, G., and Hu, X. Enhanced fixed-priority scheduling with (m, k)–firm guarantee. In *Real-time systems symposium* (2000), I. C. Society, Ed., pp. 79–88.

[8] Striegel, A., and Manimaram, G. Best-effort scheduling of (m, k)–firm real-time streams in multihop networks, 2000.